NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

NPSNET: PHYSICALLY-BASED MODELING ENHANCEMENTS TO AN OBJECT FILE FORMAT

by

James G. Monahan

September 1991

Thesis Advisor:	Michael J. Zyda
Co-Advisor:	David R. Pratt

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Naval Postgraduate School — 6b. OFFICE SYMBOL *(if applicable)*: CS/ZK	7a. NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c. ADDRESS *(City, State, and ZIP Code)*: Monterey, CA 93943-5000	7b. ADDRESS *(City, State, and ZIP Code)*: Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION — 8b. OFFICE SYMBOL *(if applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS *(City, State, and ZIP Code)*	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE *(Include Security Classification)*

NPSNET: PHYSICALLY-BASED MODELING ENHANCEMENTS TO AN OBJECT FILE FORMAT

12. PERSONAL AUTHOR(S)

Monahan, James G.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
Master's Thesis	FROM 09/89 TO 09/91	September 1991	71

16. SUPPLEMENTARY NOTATION

The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the United States Government.

17. COSATI CODES

FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*

Graphics, Animation, Physically-Based Modeling, Simulation, DoD Software Development

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

The Naval Postgraduate School (NPS) has actively explored the design and implementation of real-time three-dimensional simulators on low-cost, readily accessible graphics workstations. Many of the simulator platforms have had tremendous success due to the fact that a common object format was used. Prototyping time is dramatically reduced when the tedious and often repetitious task of object design is replaced with the simpler task of modifying an existing object description file. The current level of support that the NPS Object File Format (OFF) provides is descriptions for lights, lighting, material characteristics, the expected graphics drawing primitives (lines, polygons, surfaces,...), and provisions for texturing and special lighting effects (spotlights, decaling,...). The objectives of this research are the enhancement of the basic OFF structure with information necessary for accurate physically-based rendering in real-time; to construct a library of functions specifying an object's physical properties and the internal/external forces controlling the object and to develop a tool to rapidly design and test an object's dynamic characteristics.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: Michael J. Zyda	22b. TELEPHONE *(Include Area Code)*: (408) 646-2305 — 22c. OFFICE SYMBOL: CS/ZK

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

NPSNET: PHYSICALLY-BASED MODELING ENHANCEMENTS TO AN OBJECT FILE FORMAT

by

James G. Monahan
Lieutenant, United States Navy
B.A., Cornell University, 1983

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1991

ABSTRACT

The Naval Postgraduate School (NPS) has actively explored the design and implementation of real-time three-dimensional simulators on low-cost, readily accessible graphics workstations. Many of the simulator platforms have had tremendous success due to the fact that a common object format was used. Prototyping time is dramatically reduced when the tedious and often repetitious task of object design is replaced with the simpler task of modifying an existing object description file. The current level of support that the NPS Object File Format (OFF) provides is descriptions for lights, lighting, material characteristics, the expected graphics drawing primitives (lines, polygons, surfaces,...), and provisions for texturing and special lighting effects (spotlights, decaling,...). The objectives of this research are the enhancement of the basic OFF structure with information necessary for accurate physically-based rendering in real-time; to construct a library of functions specifying an object's physical properties and the internal/external forces controlling the object and to develop a tool to rapidly design and test an object's dynamic characteristics.

T13515
N6726
c.1

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. THE BASIC OBJECT FILE FORMAT

In the past, the various areas of real-time, three-dimensional (3D) visual simulator research at the Naval Postgraduate School (NPS) have had specific scope and purpose for a unique vehicle platform. Simulation design and implementation techniques optimized the technology of workstations in use. Advances in workstation hardware and software have always lead to more accurate simulations with each successive generation. An area of concern was to prevent an iconoclastic attitude between existing simulation projects and to facilitate the rapid prototyping of new simulator platforms. It soon became evident that future simulator development would demand a more unified protocol for object/scene description, rendering, and manipulation.

Advances in hardware capabilities such as lighting and texturing were painfully absent from these early simulations. There was no ability to quickly modify and port various objects between platforms; object renderings and control modifications were tedious for the platform's author let alone a follow-on design team. The NPS Object File Format (NPS OFF or simply OFF) initial research was designed to solve these problems by introducing an editable ASCII file with the information necessary to render an object along with various support routines to show, manipulate and save OFF objects (Zyda 1991a, Zyda 1991b).

The version 1.0 OFF consisted of lights, light model, material (color) and drawing subprimitive (lines, polygons, surfaces) definition tokens along with some administrative tokens for file maintenance and readability. The rendering of an object was accomplished in 3 steps: 1) pre-render parsing of the ASCII file into a dynamically allocated structure of object definition opcodes, 2) pre-render definition of lights and light models, 3) traversing the opcode list, drawing only the graphic primitives and selecting the "currently active" light, light model or material definition. Step 3 is the only one required each time through the display loop.

B. ADDITIONS TO THE BASIC OBJECT FILE FORMAT

Further enhancements to OFF included tokens to select textures, decaling, 2-sided lighting, spotlights and other rendering attributes. While current OFF objects *looked* just like the real world objects that they were simulating, unfortunately, many of the OFF object simulations did not *behave* realistically. As each OFF object was nothing more than a description of its "skin", it was usually animated by implicitly specifying changes in linear position/velocity and orientation. OFF objects could quite literally become "...faster than a speeding bullet, more powerful than a locomotive..." and defy many more laws of physics that we implicitly, if not explicitly, understand.

1. Incorporating Physical Realism

More recent research at NPS, specifically the Autonomous Underwater Vehicle (AUV), has taken a current OFF submarine object and animated it under the constraints of accurate hydrodynamic laws of motion (Jurewicz 1989). The result is an amazingly realistic, both visually and physically, simulation of one specific OFF object. A small drawback of the AUV simulation is that the physically-based modeling (PBM) representation of the dynamics is hardcoded. Adding/adjusting the AUV's dynamics is not a simple task, and the integration of a physically different submarine model would require software maintenance by a knowledgeable AUV programmer.

C. RELATED WORK

1. bolio

bolio is an integrated graphical simulation platform developed by David Zeltzer et al at MIT's Media Lab (Brett 1987, Sturman 1989, Zeltzer 1989). The project's goal has been to provide an environment that animates objects governed by a network of constraints (dynamic and kinematic). The bolio file format is similar in nature to OFF in that the top level file contains ASCII keyword/value pairs specifying object characteristics. While bolio identifies additional binary data structure files, OFF remains completely ASCII. The product development at NPS is almost exclusively experimental research and it was felt that a 100% human readable file format was needed during platform prototyping. When the final project design has been accepted, each OFF file can be converted into binary to decrease file I/O time.

While bolio has demonstrated exceptional realism with the constraint-based movement of a *few articulated bodies*, the OFF and the NPS simulation network (NPSNET) programs have been more concerned with the real-time animation of a *legion of 3D icons* (Zyda 1991c). Only with recent advances in workstation hardware, has there been a capability to render a multitude of minimally articulated vehicles, in real-time. The vehicular nature of most NPSNET objects has lead toward a more interactive form of object-control, rather than bolio's use of kinematically specified task-level manipulations. Also, the constraints in OFF are used more as a specification of an object's *physical capabilities*, rather than a notation for an object's *desired behavior*.

2. Virya

Virya is a graphical editor for specifying an articulated object's motion-control characteristics, designed by Jane Wilhelm's group at UCSC (Wilhelms 1986, Wilhelms 1987). A user can assign to each body's degree of freedom (DOF), one or more controlling functions (forces or torques vs. time or positions vs. time). These functions can exist in one of many control states such as position or dynamics control, frozen or relaxed. The control functions are cubic spline curves delineated by control points maintained in an ASCII file format.

3. Notion

Additional motion control work by Jane Wilhelm's group describes a technique that allows a user to depict an object's behavior based on internal sensors (provocation detectors), effectors (propulsion mechanisms) and mappings (connections and nodes) between them (Wilhelms 1990). Connections provide data transfer/ modification from sensors to effectors, while nodes permit multiple connections from many sources of input/output. This technique has been demonstrated with an interactive, workstation-based system called Notion which allows a user to specify and view an object's behavior-derived motion.

4. Dynamic Constraints

Barzel and Barr present an approach to controlling rigid bodies with dynamic constraints (Barr 1987, Barzel 1988a, Barzel 1988b). These constraints are instanced and then sustained throughout the animation using inverse dynamics. The resultant "constraint" forces determine the object's motion. Rather than construct "constraining" forces, we are more interested in specifying "controlling" forces, similar to Barzel/Barr's use of external forces to guide objects prior to constraint initiation.

D. PBM ENHANCEMENTS TO THE OBJECT FILE FORMAT

This paper describes an approach for enhancements to OFF which bestows an object with physical characteristics and provides mechanisms to govern the object's motion given a list of known internal and external forces acting on the object. We have developed a rudimentary algorithm for the automatic maintenance of multiple objects' current placement and orientation in real time. Using a tool developed at NPS called the OFF Mover Tool, a designer can view OFF objects from all perspectives, including those from an object's point of view. After a set of forces is added and adjusted in location/affect, the designer is then able to "test-drive" an object to verify its force characteristics. Constraints on the force actuators and object movement are easily added or changed. The modified OFF object is saved back to a file and is ready for integration into any simulation utilizing the OFF library of object and force functions. In Chapter II, basic dynamics theory for object animation is discussed. Chapter III describes the use of a layered approach to the creation and application of force definitions, force control and action control in OFF. Chapters IV and V describe the capabilities and performance of the NPS OFF3 Mover Tool development and testing simulator. Chapter VI concludes with a description of future work to increase the accuracy and realism of the physically-based modeling while lowering the final complexity of user-specified object movement.

II. THE DYNAMICS OF OBJECT ANIMATION

A. INTRODUCTION

The use of dynamics in rigid-body simulations requires a delicate understanding and balancing of geometric and algorithmic complexities. If we are interested in modeling the precise physical interactions of simple objects, we can afford the computational expense of dynamics simulation. Increasing an object's structural complexity and having it interact with a greater number of peer objects, strains many dynamics algorithms to the point where they are unusable for real-time simulation. It is clear that the use of dynamics to simulate Newtonian mechanics is essential for most forms of motion (ballistic, robotic, ambulatory and piloted). The following sections attempt to provide broad insight into simplifying the task of dynamics integration. Additional amplification is available in two exceptional references, Jane Wilhelm's dynamics tutorial (Wilhelms 1988) and Goldstein's mechanics theory text (Goldstein 1980).

B. NON-DEFORMING FORCES

1. Initial Conditions

The object's current position and orientation are calculated based on a set of current initial conditions:

$$\left[p_{x_0}, p_{y_0}, p_{z_0}\right] \qquad \text{(position)}$$

$$\left[\theta_{x_0}, \theta_{y_0}, \theta_{z_0}\right] \qquad \text{(orientation)}$$

$$\left[\frac{\delta}{\delta t}(p_{x_0}), \frac{\delta}{\delta t}(p_{y_0}), \frac{\delta}{\delta t}(p_{z_0})\right] \qquad \text{(linear velocity)}$$

$$\left[\frac{\delta}{\delta t}(\theta_{x_0}), \frac{\delta}{\delta t}(\theta_{y_0}), \frac{\delta}{\delta t}(\theta_{z_0})\right] \qquad \text{(angular velocity)}$$

and initial time t_0.

Often, the initial velocity values are not needed as most objects begin life motionless. Nevertheless, the ability to create an object, such as an airplane, in all phases of its movement description requires a provision for non-zero initial velocities.

2. Newton's Laws

$$F_{xyz} = m \times a_{xyz}$$

or simply "A given force acting on a given mass will accelerate it."

More specifically,

$$F_{xyz} = m \times \frac{\delta^2}{\delta t^2}(p_{xyz}) \quad \text{and} \quad T_{xyz} = m \times \frac{\delta^2}{\delta t^2}(\theta_{xyz})$$

where	F_{xyz}	= the net force directed at the object's center of mass and
	T_{xyz}	= the net torque directed at the object's center of mass
	m	= object mass
	$\delta^2/\delta t^2(p_{xyz})$	= linear acceleration component
	$\delta^2/\delta t^2(\theta_{xyz})$	= angular acceleration component

3. Local Frame of Reference

Each non-deforming force vector is converted into two collision coordinate system vectors; one that affects a torque (tangential) and one that affects a translation (radial) in each of the XY, YZ and XZ planes respectively, (Figure 2.1). Since each component of the movement force 6-vector (three tangential forces and three radial forces) is mutually exclusive, they are summed to generate a *cumulative* object frame movement force 6-vector.

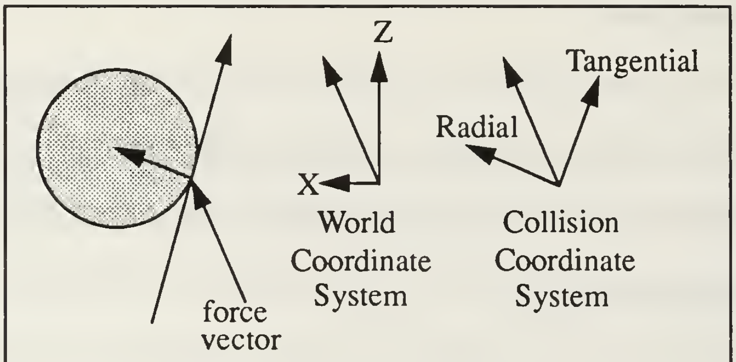


Figure 2.1 Force Components

The movement force 6-vector specifies an object-frame acceleration 6-vector (the three tangential force components create three rotation acceleration components as the remaining three radial force components create three translational accelerations). For example, in the XZ plane, only the X and Z components create linear motion and torque about the Y axis. Each force adds its effects to the object's six object frame of reference accelerations along and around each of the three object's axes. An object-frame velocity 6-vector is calculated using constant acceleration over the integration time interval δt. Both 6-vectors are then mapped into their world frame counterpart 6-vectors. These world frame accelerations and velocity 6-vectors are then used in a modified Euler integration (Spiegal 1988).

$$\frac{\delta}{\delta t}(Fp_{xyz}) = \frac{\delta}{\delta t}(Ip_{xyz}) + \left(\frac{\delta^2}{\delta t^2}(p_{xyz}) \times \delta t\right)$$

$$\frac{\delta}{\delta t}(F\theta_{xyz}) = \frac{\delta}{\delta t}(I\theta_{xyz}) + \left(\frac{\delta^2}{\delta t^2}(\theta_{xyz}) \times \delta t\right)$$

These two equations calculate final linear/angular velocities, given current velocities and accelerations over a time interval δt.

$$Fp_{xyz} = Ip_{xyz} + (\frac{\delta}{\delta t}(Fp_{xyz}) \times \delta t) + \left(0.5 \times \frac{\delta^2}{\delta t^2}(p_{xyz}) \times (\delta t)^2\right)$$

$$F\theta_{xyz} = Ip_{xyz} + (\frac{\delta}{\delta t}(F\theta_{xyz}) \times \delta t) + \left(0.5 \times \frac{\delta^2}{\delta t^2}(\theta_{xyz}) \times (\delta t)^2\right)$$

These two equations calculate final linear/angular positions, given current positions/velocities and predicted velocity averages at sample time.

where	$\delta/\delta t(Fp_{xyz})$	= final linear velocity component
	$\delta/\delta t(F\theta_{xyz})$	= final angular velocity component
	Fp_{xyz}	= final position component
	Ip_{xyz}	= initial position component
	$F\theta_{xyz}$	= final orientation component
	$I\theta_{xyz}$	= initial orientation component
	δt	= time interval since last integration

The modified Euler method was selected for its simplicity and iterative speed. Each object's force list is updated once per rendering loop, therefore nullifying the additional precision provided by second order and higher methods of integration. Obviously, Euler's method will lose accuracy as each object is subjected to rapidly changing forces. A future implementation will sample the force updates in parallel, track the relative changes in linear/angular accelerations and switch to a higher order integration method, such as Runge-Kutta, under a rapidly moving scenario.

4. Global Frame of Reference

Each global force (such as gravity) affects the object at its center of mass causing only linear acceleration. Since the movement does not involve rotations, it can be added after the net effect of all local forces is determined.

C. DEFORMING FORCES

A deforming force affects the object in one of three ways. Each polygon in the object has an associated *break* and *bend threshold* token specified in newtons/meter^2. Using the relationship that a force dissipates its kinetic energy inversely over the square of the distance from the force origin to the polygon, a dissipated force per unit polygon surface area value is calculated. If the force is strong enough to break the polygon, the original polygon token is removed from the object token list and replaced with a list of smaller triangular polygonal shard tokens, (Figure 2.2). Triangles are used to guarantee planar polygons.

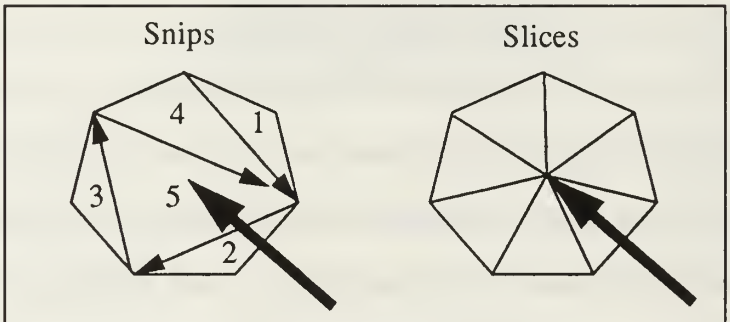


Figure 2.2 Breaking Force

The shards are initially determined by "snipping" off the corners of a multi-sided convex polygon, thus spiraling inward until the remaining quadrangle is divided in two. The rationale is to 1) prevent identical "pizza slice" shards as explosions are rarely symmetrical and 2) generate (*n*-2) versus *n* fragments from an *n*-sided polygon. Any remaining shards are broken along their hypotenuse, as needed.

If the force is only strong enough to bend the polygon, the polygon token is removed from the object token list and replaced with a new bendable polygon that tracks a moving point of bending force impact, (Figure 2.3). The bending force is modeled using Hooke's

Law and a spherical spring that seeks to return the moving vertex back to the polygon's actual point of impact.

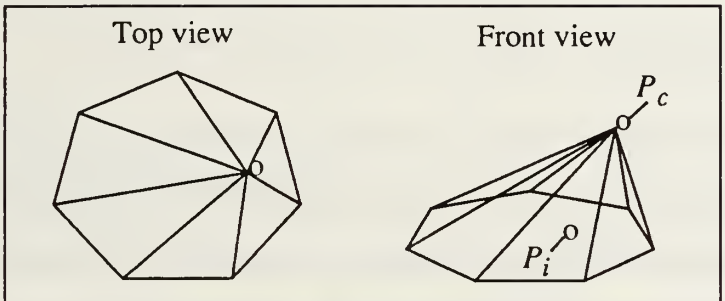


Figure 2.3 Bending Force

$$F_{xyz} = -(k_s \times \Delta p_{xyz} + k_d \times \Delta v_{xyz})$$

where Pi	= initial point of bending force impact
Pc	= current point of bending force impact
F_{xyz}	= linear bending component
k_s	= spring constant
k_d	= damping constant
Δp_{xyz}	= difference between the position components of the initial and current points of impact
Δv_{xyz}	= difference between the velocity components of the initial and current points of impact

If the force is neither strong enough to break or bend a polygon, then it may only push a polygonal shard.

III. A LAYERED APPROACH

A. OBJECTIVE

The initial objective of this research was to provide a structured mechanism for object behavior control that would allow varying degrees of *user* and *designer* involvement with the simulation. The end user is concerned with realism: visual accuracy and similarity of interface (i.e *look* and *feel*). The simulated object's appearance and movement must closely resemble its real world counterpart. Objects that instantly accelerate to highly unnatural velocities provide a temporary sensation of giddiness, bordering upon the comical. A lack of expected visual clues from the objects' Newtonian interactions, such as the lack of gravity effects, reduces the allure of the virtual reality. A similar degradation in simulator immersion is also evident when the user is expected to identify and interface with an inordinate number of forces, which are manipulating several objects.

B. METHOD

The desired implementation would allow the art design team to create a specific object hierarchy with realistic shape, colorings and positioning data. The engineering section would then add the subobject physical attributes (mass, center of mass and object elasticity) and affecting force descriptions (force position, or point of affect in the object's frame of reference, along with the force direction unit vector, magnitude and type of force). Reasonable defaults for omitted physical attributes are assumed and/or calculated from other specifications. The analysis team would then specify mappings between subobject movement and the forces affected by such movement. The end user is then able to control a given object in a realistic manner with realistic results by manipulating a set of *control subobjects* linked to local forces. The result is an adjustable "focus" in specifying high-level object motion in a range of control modes: directly, indirectly through local *force* control, and even more indirectly with *subobject* control.

The final objective was to design a suite of tools, so that a single user, with even a limited background in Newtonian mechanics, could rapidly design and test an object's physical characteristics.

C. DESCRIPTIONS OF EACH LAYER

Similar to Barr's view of Teleological Modeling (Barr 1988), the approach is for each layer to provide a control description to the layer below. The lowest layer consists of rendering descriptions (drawing primitives, materials and lighting controls to color their skins). The second layer consists of the object's physical characteristics and a set of force descriptions (a list of forces and their influence upon specific objects). The third layer consists of action descriptions (a mapping of an object's movements to changes in a set of force descriptions) (Figure 3.1).

D. PRIMITIVE ADDITIONS

As described in Chapter I, the version 1.0 OFF file supported only drawing subprimitives such as lines, polygons, and meshed surfaces. Other common generalized surface primitives (cones, cylinders, spheres and parallelepipeds) were usually calculated off-line, with their polygonal data stored into an OFF file. The major disadvantage to this approach was that these OFF files were extremely large and difficult to edit. As scenarios requiring different colors on a primitive were rare at best, the inclusion of a set of parametrized primitive descriptions was required. For example, a cylinder token is specified by a height, radius, and quality factor that indicates the maximum number of polygons or mesh points to use in the rendering. The advantages were automatic minimal polygonal computation based on ranging data from a specified viewpoint, a simple

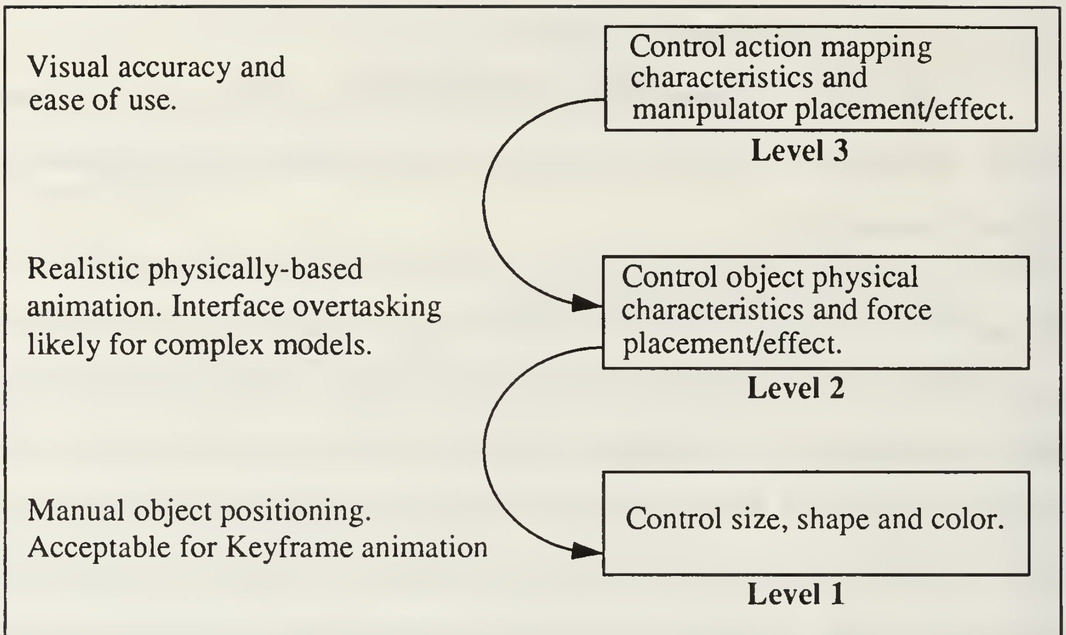

Visual accuracy and ease of use.
Control action mapping characteristics and manipulator placement/effect.
Level 3
Realistic physically-based animation. Interface overtasking likely for complex models.
Control object physical characteristics and force placement/effect.
Level 2
Manual object positioning. Acceptable for Keyframe animation
Control size, shape and color.
Level 1

Figure 3.1 Layered Approach to Control Modeling

mechanism for multiple object resolution creation and known mass/center of mass values, to name a few.

E. OBJECTS AND FORCE CONTROL

The models in the various NPS simulators have quite an eclectic background. Some came from non-organic sites such as NASA and MIT, requiring conversion from other file formats. Many others were designed inhouse and more often than not, the various models were rarely scale compatible. The first set of extensions to the OFF language, (Table 1), included tokens to specify and convert between the various units of measure (Layer 1).

TABLE 1: UNITS OF MEASURE

Token Function	Argument Type(s)
units of dimension	char
units of force	char
units of mass	char

1. Object Modeling Requirements

The next set of extensions, (Table 2), included tokens to specify the physique, initial conditions, and motion boundary conditions of an object (Layer 2).

TABLE 2: OBJECT CHARACTERISTICS

Token Function	Argument Type(s)
initial position	float, float, float
position constraints (low)	float, float, float
position constraints (high)	float, float, float
initial rotation	float, float, float
rotation constraints (low)	float, float, float
rotation constraints (high)	float, float, float
initial linear velocity	float, float, float
linear velocity constraints (low)	float, float, float
linear velocity constraints (high)	float, float, float
initial rotation velocity	float, float, float
angular velocity constraints (low)	float, float, float
angular velocity constraints (high)	float, float, float
mass	float
center of mass	float, float, float
elasticity	float
bounding volume radius	float
bounding volume length	float
bounding volume width	float
bounding volume height	float
viewpoint from object	float, float, float

A default bounding volume is calculated as the object is read into memory. Provisions for specifying a smaller (or larger) bounding description (spherical, rectangular or ellipsoid) were added to facilitate parallel research efforts in collision detection. As much of the research at NPS involves the simulation of piloted vehicles, the inclusion of a vehicle viewpoint was a requirement.

2. Force Modeling Requirements

The next set of extensions, (Table 3), included tokens to specify the capabilities and constraints of a force acting upon an object (Layer 2).

TABLE 3: FORCE CHARACTERISTICS

Token Function	Argument Type(s)
name	char
type	deforming, non-deforming, or global
origin	float, float, float
origin constraints (low)	float, float, float
origin constraints (high)	float, float, float
direction	float, float, float
magnitude	float
magnitude constraints	float, float
asleep	yes or no

The concept of a "sleeping" or suspended force is to leave the force attached to the object but remove its effect. The rationale for this was for simplification during the force definition and analysis phase. A given force can be isolated by leaving it the sole "awake" or active force. The unacceptable alternative is to nullify the other forces by restricting their magnitudes and/or directions.

F. OBJECTS AND ACTION CONTROL

What we have now is a "marionette" object that is manipulated by pulling and pushing "force" strings and rods. For objects with a simple description of force effectors, this is quite acceptable as each force can be visualized as a controlling "object" rather than a force. Realistically, most objects' movements are *described* by a complex network of forces, yet *controlled* from a small number of input sources. Layer 3's objective will be to identify a small set of *control* objects and provide a mapping from their movement (translations/rotations) changes to a set of force description (origin/direction/magnitude) changes.

We will want to specify a set of *dynamic* controlling forces and then provide an abstraction for altering their affect based on user input changes. As early key-frame animation control systems identified key object positions and then interpolated the in-between positions as a function of time, so should we specify key controlling force effects and then interpolate the in-between effect components as functions of a user input position/orientation. This approach will provide for a natural migration from *simulated* input sources to *actual hardware/sensor* input sources.

For example, a jet object has three forces that describe the effects of two ailerons and one stabilizer. The action induced by their movement is controlled by one input source, the pilot's stick. The user will describe simple mappings for the stick's lateral rotation (affecting the aileron forces) and longitudinal rotation (affecting the stabilizer force). Adding additional mappings for throttle/rudder pedal positions and we will have an airplane that is fully controllable with changes in the *input device's position and orientation*, (Figure 3.2).

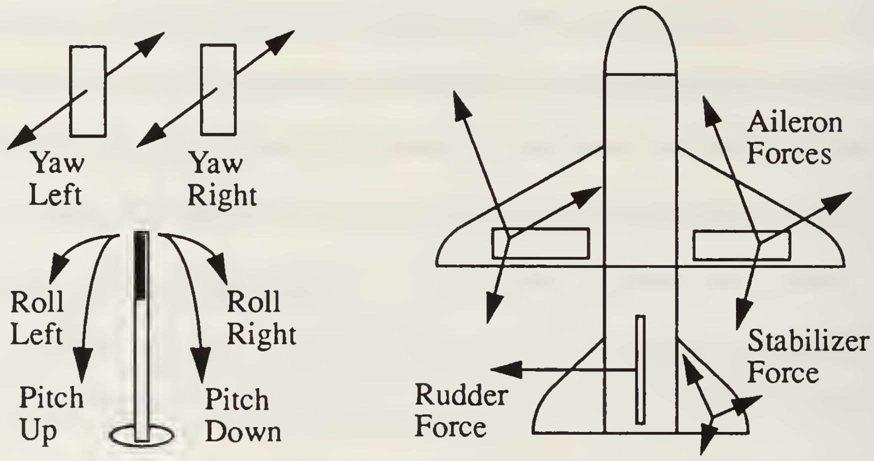


Figure 3.2 Linking Subobject Movement to Forces

As the construction of Layer 3 is part of continuing research, the following is a *specification* for a possible OFF file mapping from subobject movement (translations/ rotations) changes to a set of force description (origin/direction/magnitude) changes:

```
defmapping sample_map_name
    sample_object             sample_object_name
    sample_force              sample_force_name
    rot_to_force_origin       matrix
    rot_to_force_vector       matrix
    trans_to_force_origin     matrix
    trans_to_force_vector     matrix
defend
```

Each object and force has an initial (neutral) state specified in their respective descriptions. Each respective 3x3 matrix would transform a 3-vector (controlling object rotation and translation changes) into another 3-vector (force origin and vector incremental updates). Additional mappings would require velocity information as well.

IV. THE OFF MOVER TOOL

A. MOTIVATION

The objective of the OFF Mover Tool is to provide an environment to design and test the dynamics of OFF objects. An OFF object without physical characteristics is read into memory from disk and the object is measured for future calculations. A default mass, mass center, elasticity and object viewpoint are calculated. The user is then able to "fine tune" any of these approximations based on known data. The user then specifies the initial values for object position, orientation and velocity. At the lowest layer of the tool's control, the user is able to continually update the object's movement by indicating the linear/angular direction and speed. This would be acceptable for specifying instances for a keyframing sequencer, but we are more interested in providing mechanisms to accelerate the object just like its real world counterpart. The mechanism of choice is a force description.

B. APPLICATION

The user controls a set of forces that are in turn, controlling the object's movement. A force is positioned around an object and its range of effect is specified. For example, our jet fighter object is re-read into the Mover Tool and the engine forces are added via a force interface, (Figure 4.1). A separate force vector is positioned in the center of each exhaust nozzle, initially directed forward (direction of the *reactive* force) and parallel to the turbine housing, with a zero newton magnitude. The force's magnitude is constrained by a non-negative range of thrust values. The point of effect is also given a small range of values along the axis parallel to the direction of thrust to account for the change in thrust position when the engine is operated in afterburner and additional fuel is combusted behind the main

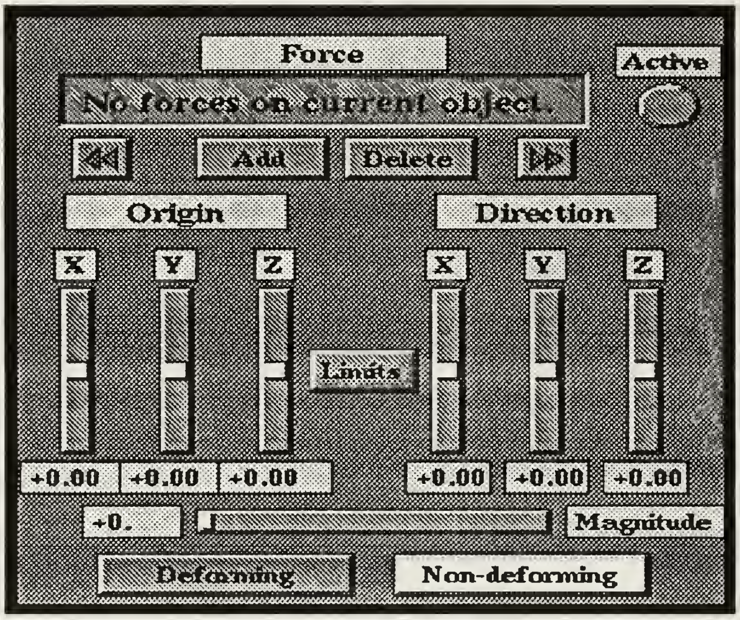


Figure 4.1 Force Panel

combustion chamber. Similar forces are added to the wings for lift and drag forces created by the various control surfaces (flaps, ailerons, spoilers,...).

Additional pseudo forces, such as parasitic drag can be added to tune the realism. The result is a vehicle that will maneuver with amazing realism. The object can be "test flown" in isolation or with other testbed objects to verify the object's force parameters. The object is saved to a file when the desired physical model is accepted. The OFF file force descriptions are always editable with any ASCII text editor.

The following is a sample OFF file force description for the plane's left engine:

```
defforce left_jet_engine
   force_type                        non-deforming
   force_origin                      -4.0 0.5 -0.8
   force_origin_low                  0.0 0.0 0.0
   force_origin_high                 -4.5 0.5 -0.8
   force_direction                   -1.0 0.0 0.0
   force_magnitude                   8000.0
   force_magnitude_constraints       0.0 10000
   asleep                            no
defend
```

C. POSTDESIGN

The same functions that are used to animate objects in the OFF Mover Tool are embedded in the NPS OFF function library. In addition to the functions that read in an object file, ready it for display, and display it each time through the display loop, are a host of new functions that: add/delete objects and forces from the animation environment, navigate the object/force lists, alter the object/force parameters, and start/stop the animation process. For more detail, see Appendix B: PHYSICALLY-BASED PROCEDURES.

V. OFF SAMPLES AND INTEGRATION

A. OFF FILE SAMPLE

The following is a fragment of an OFF file description of an SU-25 Frogfoot Soviet ground attack aircraft.

```
/* These are ALL of the required units of measure. */
        Note: each subtoken is separate and none are required (defaults used). */
defunits
    /* All lengths are in meters. Other length choices are available. */
    dimension meters
    /* All force magnitudes are in newtons. Other force choices are available. */
    force newtons
    /* All mass amounts are in kilograms. Other mass choices are available. */
    mass kilos
defend

/* These are ALL of the required object characteristics.
        Note: each subtoken is separate and none are required (defaults used). */
defphysics
    /* This object's initial position is (X,Y,Z) in meters relative from the
    environments's center. Unless otherwise specified, all triples are X,Y,Z
    respective. */
    location 0.00 0.00 0.00

    /* The object's position is constrained to a one meter level square, relative from
    the object's initial position. */
    location_lower -1.00 -0.00 -1.00
    location_upper 1.00 0.00 1.00

    /* This object's initial orientation (Roll, Yaw, Pitch) in degrees. */
    orientation 0.00 0.00 0.00

    /* The object's orientation is unconstrained. */
    orientation_lower 0.00 0.00 0.00
    orientation_upper 360.00 360.00 360.00

    /* The object's initial linear velocity in meters/second. */
    linear 0.00 0.00 0.00
    /* The object's linear velocity is constrained to: 0.00 to 1000.0 longitudinal,
     +/- 1000.0 vertical and +/- 500.0 latitudinal. */
    linear_lower 0.00 -1000.00 -500.00
    linear_upper 1000.00 1000.00 500.00

    /* The object's initial angular velocity in degrees/second. */
    angular 0.00 0.00 0.00

    /* The object's angular velocity is constrained to: +/- 10.00 longitudinal, vertical
    and latitudinal. */
    angular_lower -10.00 -10.00 -10.00
    angular_upper 10.00 10.00 10.00

    /* The object's center of mass and amount in kilos. */
    mass_amount 25000.00
    mass_center 0.00 0.00 0.00

    /* The object's ability to absorb local forces. (0.0 is perfectly inelastic) */
    elasticity 0.80

    /* The dimensions of the object's bounding volume (e.g. for collision detection).
    The volume dimensions are calculated if this data is omitted. */
    bv_radius 30.00
    bv_latitude 15.00
    bv_longitude 20.00
    bv_vertical 8.0

    /* The location of the object's local viewpoint.
    setviewpoint 0.00 38.241650 0.00
defend

/* These are ALL of the required force characteristics for this force.
        Note: each subtoken is separate and none are required (defaults used). */
defforce left_jet_engine
    force_type non-deforming
    force_origin -4.0 0.5 -0.8
    force_origin_low 0.0 0.0 0.0
    force_origin_high -4.5 0.5 -0.8
    force_direction -1.0 0.0 0.0
    force_magnitude 8000.0
    force_magnitude_constraints 0.0 10000
    asleep no
defend

/* Additional forces (right_engine, left_aileron,...) would follow here. */
/* The next two definitions specify a polygon lighting/shading characteristic.
      Note: each subtoken is separate and none are required (defaults used). */
defmaterial su25mat0
   emission 0.00 0.00 0.00
   ambient 0.047059 0.086275 0.047059
   diffuse 0.235294 0.431373 0.235294
   specular 0.00 0.00 0.00
   shininess 0.00
   alpha 1.00
defend

defmaterial su25mat1
   emission 0.00 0.00 0.00
   ambient 0.047059 0.094118 0.047059
   diffuse 0.235294 0.470588 0.235294
   specular 0.00 0.00 0.00
   shininess 0.00
   alpha 1.00
defend

/* The remaining defmaterials go here. */


/* A particular lighting/shading characteristic is activated. */
setmaterial su25mat0

/* The next definition specifies a triangular polygon.
      Note: each line is NOT separate and ALL are required. */
defpoly
   /* Normal, number of vertices and vertex coordinates. */
   0.875439 -0.483329 0.00
   3
   40.142231 6.476233 1.029732
   39.087486 4.565808 -1.076130
   40.142231 6.476233 -1.029732

/* The remaining primitive definitions go here. */
```

B. INTEGRATION SAMPLE

The following code fragments demonstrate the various phases of OFF file integration with the force/object functions. For more detail, see Appendix B: PHYSICALLY-BASED PROCEDURES.

1. Initializing The Environment

```
initialize_environment();

/* Let's add gravity. */
add_global_force();
strcpy(current_global_forceptr->name,"gravity");
modify_force_origin(current_global_forceptr,0.0,1.0,0.0);
modify_force_direction(current_global_forceptr,0.0,-1.0,0.0);
```

2. Adding And Modifying An Object

```
objectptr = read_object("sample_filename");
ready_object_for_display(objectptr);
add_object_to_environment(objectptr);

/* Any characteristics (specified or not in the OFF file) can be modified.
We can check if a particular object characteristic has changed (e.g. by polling
an input device), add adjust it prior to calculating the objects' motion. */
modify_object_position(objectptr,px,py,pz);
modify_object_position_lower(objectptr,lx,ly,lz);
modify_object_position_upper(objectptr,ux,uy,uz);
modify_object_rotation(objectptr,rx.ry,rz);
modify_object_rotation_lower(objectptr,lx,ly,lz);
modify_object_rotation_upper(objectptr,ux,uy,uz);
modify_object_linear_velocity(objectptr,vx,vy,vz);
modify_object_linear_velocity_lower(objectptr,lx,ly,lz);
modify_object_linear_velocity_upper(objectptr,ux,uy,uz);
modify_object_angular_velocity(objectptr,vx,vy,vz);
modify_object_angular_velocity_lower(objectptr,lx,ly,lz);
modify_object_angular_velocity_upper(objectptr,ux,uy,uz);
modify_object_mass(objectptr,mass,mx,my,mz);
modify_object_bounds(objectptr,radius,latitude,longitude,vertical);

/* If the object needs to be removed, we delete it. */
delete_object_from_environment(objectptr);

/* If we want to suspend all forces on a particular object, */
suspend_object(objectptr);

/* or to re-allow all active forces to influence the object. */
wakeup_object(objectptr);
```

3. Adding And Modifying A Force

```
add_local_force(objectptr);
/* or */
add_global_force();
```

```
/* Any characteristics (specified or not in the OFF file) can be modified. */
We can check if a particular force characteristic has changed (e.g. by polling
an input device), add adjust it prior to calculating the objects' motion. */
modify_force_origin(forceptr,ox,oy,oz);
modify_force_origin_lower(forceptr,lx,ly,lz);
modify_force_origin_upper(forceptr,ux,uy,uz);
modify_force_direction(forceptr,ox,oy,oz);
modify_force_magnitude(forceptr,magnitude);
modify_force_magnitude_constraints(forceptr,lower,upper);
modify_force_type(forceptr,type);

/* If the force needs to be removed, */
delete_local_force(objectptr,forceptr);
/* or */
delete_global_force(forceptr);

/* If we want to suspend a force, */
suspend_force(forceptr);
/* or to re-allow this force to influence the object. */
wakeup_force(forceptr);
```

4. Updating The Object's Physics

The following is a simple example of a code fragment to traverse the list of all forces attached to all objects. Parallelization of the force computations per object is part of continuing research.

```
/* olist is an object list global variable. action_objectptr is a local variable that
points to an object*/
action_objectptr = olist.first;

/* For each object in the environment, */
while (action_objectptr != (OBJECT *) NULL) {
/* If some time has elapsed and the object is awake, */
if ((action_objectptr->last_time > 0.0) && (!action_objectptr->asleep)) {
    /* Calculate the time since the object was last moved. */
    deltatime = delta_time(&(action_objectptr->last_time));
    /* Zero the previous summation force 6-vector. */
    for (i=0; i<3; i++) {
        summoveptr->rotation[i] = 0.0;
        summoveptr->translation[i] = 0.0;
    }
/* action_forceptr is a local variable that points to a force*/
    action_forceptr = action_objectptr->firstforceptr;
/* For each local force attached to this object, */
    while (action_forceptr != (FORCE *) NULL) {
        /* If this force deforms and is awake, */
        if ((action_forceptr->type == DEFORMING) &&
            (!action_forceptr->asleep)) {
            /* If the object is not already exploding, apply the force. */
            if (!action_objectptr->exploding_flag)
                apply_deforming_force_to_object(action_objectptr,
                        action_forceptr, deltatime);

            else
            continue_explosion(action_objectptr,deltatime);
        }
        /* If this force does not deform and is awake, apply it. */
        else if ((action_forceptr->type == NON_DEFORMING) &&
                (!action_forceptr->asleep))
                apply_non_deforming_force_to_object(action_objectptr,
                    action_forceptr, deltatime);
        /* Continue with the next force. */
        action_forceptr = action_forceptr->next;
    }
    action_forceptr = glist.first;
/* For each global force, */
    while (action_forceptr != (FORCE *) NULL) {
        apply_global_force_to_object(action_objectptr,
                        action_forceptr, deltatime);
        /* Continue with the next force. */
        action_forceptr = action_forceptr->next;
    }
    update_object_physics(action_objectptr,deltatime);
}
/* Otherwise, a δt has not passed. */
else action_objectptr->last_time = now_time();

/* Continue with the next object. */
action_objectptr = action_objectptr->next;
}
```

VI. PERFORMANCE

The following tables are used to illustrate the cost associated with this physically-based modeling technique. Test case group A involves a small, average and large polygon count object with small, average and large non-deforming force lists (Table 4). Test case group B involves small, average and large sets of an average polygon count object with small, average and large non-deforming force lists (Table 5). Test case group C involves a small, average and large polygon count object with a small deforming force list, during the explosion phase (Table 6). All numbers are in frames/second.

TABLE 4: OBJECT SIZE VERSUS NUMBER OF FORCES

	Small **Polygon Count** (6) Object	*Average* **Polygon Count** (165) Object	*Large* **Polygon Count** (960) Object
Single **Non-deforming** Force	17.117	16.547	11.174
Small **Set of Non-deforming** Forces (5)	20.083	18.166	11.864
Medium **Set of Non-deforming** Forces (10)	19.923	18.063	11.519
Large **Set of Non-deforming** Forces (20)	19.525	17.876	10.765

TABLE 5: NUMBER OF OBJECTS VERSUS NUMBER OF FORCES

	Average Number of Objects (5)	*Large* Number of Objects (10)
Small Set of Non-deforming Forces (5)	11.861	8.441
Average Set of Non-deforming Forces (10)	11.525	8.179
Large Set of Non-deforming Forces (20)	10.935	8.025

TABLE 6: OBJECT SIZE VERSUS A DEFORMING FORCE

	Small Polygon Count (6) Object	*Average* Polygon Count (165) Object	*Large* Polygon Count (960) Object
Small Set of Deforming Forces	16.535	15.493	10.454

A note of interest in case A - the frame rate actually increases from one force to five forces and then decreases from then on. We are reclaiming idle CPU time and improving graphics-CPU overlap.

In the case of non-deforming forces, the frame rate decreases linearly with the *total number of non-deforming forces* attached to all objects, (Figures 5.1 and 5.2). In the case of deforming forces, the frame rate decreases linearly with the *number of initial polygons* in the pre-destroyed object (Figure 5.3).

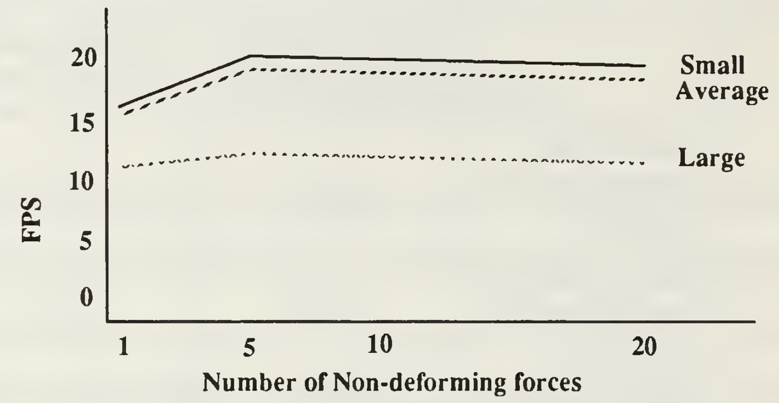


Figure 5.1 Object Size and Non-deforming Forces

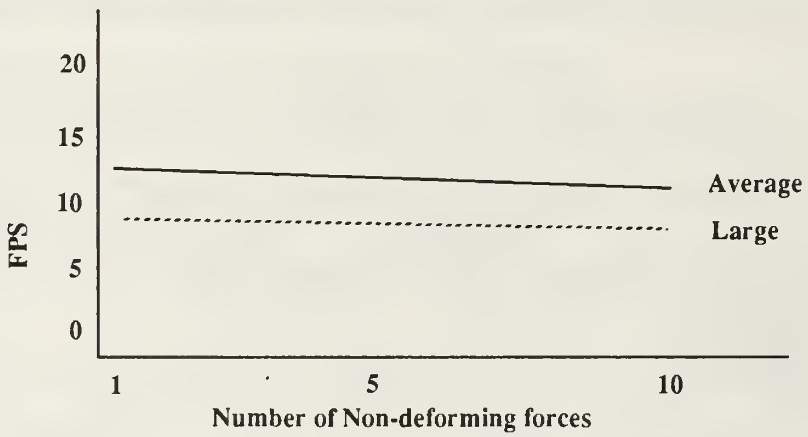


Figure 5.2 Number of Objects and Non-deforming Forces

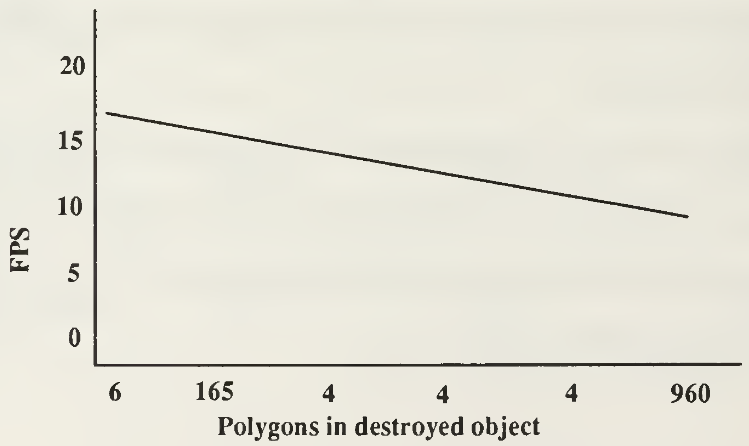


Figure 5.3 Object Size and Deforming Forces

VII. CONCLUSIONS AND FUTURE WORK

The use of physically-based modeling is still in its infancy at NPS. Previous simulations were able to "fake" or "downplay" the expected visual clues from an object's physical interactions. As hardware and software technology afford us with greater *capability* in animation realism, we are obligated to strive for more accurate physical modeling, but not at the expense of increased *user workload* to specify and control the animation process. The extensions to OFF present a simplified mechanism for building models with physical characteristics and adding controlling functions that are as complex as necessary given the current hardware support.

Future work includes the implementation of the ACTION CONTROL layer using the specification in Chapter III. Generation of the mapping function matrices could be achieved quite easily by selecting the object/force pair and then taking "snapshots" of a series of object motion/force description couplings. Each coupling would then be displayed in a 2D graph (object component vs. force component) for any desired function smoothing/ modification. Addition, deletion, and modification mechanisms would function similarly to identical object functions in key-framing systems.

Further refinements to the integration process would include parallelization of the force sampling process and the addition of an adaptive algorithm for more accurate positioning of objects with rapidly fluctuating forces.

APPENDIX A INTEGRATING OBJECTS INTO EXISTING PROGRAMS

A. OFF TOKEN ADDITIONS

```
#define DEFUNITSTOKEN 200
#define DIMENSIONUNITSTOKEN 201
#define FORCEUNITSTOKEN 202
#define MASSUNITSTOKEN 203

#define DEFPHYSICSTOKEN 210
#define LOCATIONTOKEN 211
#define ORIENTATIONTOKEN 212
#define LINEARVELOCITYTOKEN 213
#define ANGULARVELOCITYTOKEN 214
#define MASSAMOUNTTOKEN 215
#define MASSCENTERTOKEN 216
#define ELASTICITYTOKEN 217
#define BVRADIUSTOKEN 218
#define BVLATITUDETOKEN 219
#define BVLONGITUDETOKEN 220
#define BVVERTICALTOKEN 221
#define SETVIEWPOINTTOKEN 222

#define DEFBENDTOKEN 230
#define DEFBREAKTOKEN 231
#define DEFSPRINGTOKEN 232
#define DEFMASSPERSATOKEN 233

#define SETBENDTOKEN 240
#define SETBREAKTOKEN 241
#define SETMASSPERSATOKEN 242
#define SETSPRINGTOKEN 243
#define SETGRANULARITYTOKEN 244
#define DEFBENDPOLYTOKEN 250
```

```
#define DEFFORCETOKEN 260
#define FORCETYPETOKEN 261
#define FORCEORIGINTOKEN 262
#define FORCEDIRECTIONTOKEN 263
#define FORCEMAGNITUDETOKEN 264
#define FORCEASLEEPTOKEN 265

#define DEFBALLTOKEN 270
#define DEFCONETOKEN 271
#define DEFCYLINDERTOKEN 272
#define DEFBOXTOKEN 273

#define RADIUSTOKEN 280
#define PANELSTOKEN 281
#define MODETOKEN 282
#define HEIGHTTOKEN 283
#define WIDTHTOKEN 284
#define LENGTHTOKEN 285
#define LOCATIONLOWERCONSTRAINTSTOKEN 290
#define LOCATIONUPPERCONSTRAINTSTOKEN 291
#define ORIENTATIONLOWERCONSTRAINTSTOKEN 292
#define ORIENTATIONUPPERCONSTRAINTSTOKEN 293
#define LINEARVELOCITYLOWERCONSTRAINTSTOKEN 294
#define LINEARVELOCITYUPPERCONSTRAINTSTOKEN 295
#define ANGULARVELOCITYLOWERCONSTRAINTSTOKEN 296
#define ANGULARVELOCITYUPPERCONSTRAINTSTOKEN 297

#define FORCEORIGINLOWERCONSTRAINTSTOKEN 300
#define FORCEORIGINUPPERCONSTRAINTSTOKEN 301
#define FORCEMAGNITUDECONSTRAINTSTOKEN 302
```

B. SUBPRIMITIVE DATA STRUCTURES USED

```
struct defpoly {
long ncoords;
float nrml[3];
float deltap[3];
float velocity[3];
int explodingflag,
bendingflag;
float *xyz;
};
typedef struct defpoly DEFPOLY;

struct defbend {
char *name;
float bendvalue;
};
typedef struct defbend DEFBEND;

struct defbreak {
char *name;
float breakvalue;
};
typedef struct defbreak DEFBREAK;

struct defspring {
char *name;
float constant;
float minimum;
};
typedef struct defspring DEFSPRING;

struct defmasspersa {
char *name;
float masspersa;
};
typedef struct defmasspersa DEFMASSPERSA;
```

```
struct setgranularity {
float value;
};
typedef struct setgranularity SETGRANULARITY;
```

C. SUBPRIMITIVE DATA STRUCTURES USED

```
struct defball {
float radius;
float panels;
int mode;
};

typedef struct defball DEFBALL;

struct defcone {
float radius;
float panels;
float height;
};
typedef struct defcone DEFCONE;

struct defcylinder {
float radius;
float panels;
float height;
};
typedef struct defcylinder DEFCYLINDER;

struct defbox {
float length;
float width;
float height;
};
typedef struct defbox DEFBOX;
```

D. OBJECT PHYSICS DATA STRUCTURES USED

```
struct units {
char *forceunits;
char *massunits;
char *dimensionunits;
float forceconversion;
float massconversion;
float dimensionconversion;
};
typedef struct units UNITS;

struct physics {
float mass;
float massxyz[3];
float elasticity;
float acceleration[6];
float velocity[6];
float velocity_constraints[2][6];
float position[3];
float position_constraints[2][3];
float rotation[3];
float rotation_constraints[2][3];
float bvradius;
float bvlatitude;
float bvlongitude;
float bvvertical;
float viewpoint[3];
};
typedef struct physics PHYSICS;

struct force {
char *name;
int type;
float origin[3];
float origin_constraints[2][3];
float direction[3];
float magnitude;
float magnitude_constraints[2];
int asleep;
struct force *next;
struct force *prev;
};
typedef struct force FORCE;

struct movement {
float rotation[3];
float translation[3];
};
typedef struct movement MOVEMENT;
struct opcode {
long whattype;
struct opcode *prev;
struct opcode *next;
union dptr data;
};
typedef struct opcode OPCODE;

struct obj {
struct opcode *first;
struct opcode *last;
char *name;
float first_origin[3];
long origin_set;

UNITS *unitsptr;
PHYSICS *physicsptr;
FORCE *firstforceptr;
FORCE *lastforceptr;

float last_time;
int asleep;
int bending_flag,
exploding_flag;

struct obj *next;
struct obj *prev;
};
typedef struct obj OBJECT;

struct objectlist {
OBJECT *first;
OBJECT *last;
};
typedef struct objectlist OBJECTLIST;

struct globalforcelist {
FORCE *first;
FORCE *last;
};
typedef struct globalforcelist GLOBALFORCELIST;
```

APPENDIX B PHYSICALLY-BASED PROCEDURES

A. ENVIRONMENT FUNCTION SPECIFICATIONS

void **initialize_environment**()

Creates the environment (lacking objects/forces). Initializes the current object, local and global forces.

void **add_object_to_environment**(Object*)

Adds the specified object to the environment. This object becomes the current object.

void **delete_object_from_environment**(Object*)

Deletes the specified object from the environment. Returns any allocated memory assigned/linked to this object. The first, last and current objects are adjusted accordingly.

B. OBJECT FUNCTIONS

OBJECT ***duplicate_object**(Object*)

Creates another object in memory with identical information to the specified object. Useful prior to when you want to destroy an object, thus keeping a undamaged object intact.

void **suspend_object**(Object*)

Suspends the specified object so that it is no responding to any force inputs.

void **wakeup_object**(Object*)

Wakes up the specified object so that it will respond to all active force inputs.

void **modify_object_position**(Object*, float px, py, pz)

Alters the specified object's X, Y, and Z position components.

void **modify_object_position_lower**(Object*, float lx, ly, lz)

Alters the specified object's X, Y, and Z position lower boundary values.

void **modify_object_position_upper**(Object*, float ux, uy, uz)

Alters the specified object's X, Y, and Z position upper boundary values.

void **modify_object_rotation**(Object*, float rx.ry, rz)

Alters the specified object's X, Y, and Z rotation components.

void **modify_object_rotation_lower**(Object*, float lx, ly, lz)

Alters the specified object's X, Y, and Z rotation lower boundary values.

void **modify_object_rotation_upper**(Object*, float ux, uy, uz)

Alters the specified object's X, Y, and Z rotation upper boundary values.

void **modify_object_linear_velocity**(Object*, float vx, vy, vz)

Alters the specified object's X, Y, and Z linear velocity components.

void **modify_object_linear_velocity_lower**(Object*, float lx, ly, lz)

Alters the specified object's X, Y, and Z linear velocity lower boundary values.

void **modify_object_linear_velocity_upper**(Object*, float ux, uy, uz)

Alters the specified object's X, Y, and Z linear velocity upper boundary values.

void **modify_object_angular_velocity**(Object*, float vx, vy, vz)

Alters the specified object's X, Y, and Z angular velocity components.

void **modify_object_linear_velocity_lower**(Object*, float lx, ly, lz)

Alters the specified object's X, Y, and Z angular velocity lower boundary values.

void **modify_object_linear_velocity_upper**(Object*, float ux, uy, uz)

Alters the specified object's X, Y, and Z angular velocity upper boundary values.

void **modify_object_mass**(Object*, float mass, mx, my, mz)

Alters the specified object's mass and mass center.

void **modify_object_bounds**(Object*, float radius, latitude, longitude, vertical)

Alters the specified object's bounding volume as described by radius, width, length and height.

void **stop_object**(Object*)

Resets all of the specified object's velocities.

C. GENERAL FORCE FUNCTIONS

void **add_local_force**(Object*)

Adds a default local force to the end of the specified object's force list.

void **delete_local_force**(Object*, force*)

Deletes the specified force from the specified object's force list. Returns the memory allocated to the force. The first, last and current local forces are adjusted accordingly.

void **delete_global_force**(Force*)

Deletes the specified global force from the global force list. Returns the memory allocated to the force. The first, last and current global forces are adjusted accordingly.

void **suspend_force**(Force*)

Suspends the specified force so that it will not affect the object to which it is attached.

void **wakeup_force**(Force*)

Wakes up the specified force so that it will affect the object to which it is attached.

void **modify_force_origin**(Force*, float ox, oy, oz)

Alters the specified force's X, Y, and Z origin components.

void **modify_force_origin_upper**(Force*, float ux, uy, uz)

Alters the specified force's X, Y, and Z origin upper boundary values.

void **modify_force_direction**(Force*, float ox, oy, oz)

Alters the specified force's X, Y, and Z direction components.

void **modify_force_magnitude**(Force*, float magnitude)

Alters the specified force's magnitude.

void **modify_force_magnitude_constraints**(Force*, float lower, upper)

Alters the specified force's magnitude lower and upper boundary values.

void **modify_force_type**(Force*, Char* type)

Alters the specified force's type (DEFORMING, NON_DEFORMING,GLOBAL)

D. DEFORMING FORCE FUNCTIONS

void **poly_midpoint**(defpoly*, middle)

void **text_poly_midpoint**(defpolyt*, float* middle)

void **line_midpoint**(float* A, B, middle) void **text_line_midoint**(float* A, B, middle)

float **euclid**(float* A, B)

Returns the Euclidean distance between the two float triples.

float **text_euclid**(float* A, B)

Returns the Euclidean distance between the two float triples.

float **text_triangle_area**(float* A, B, C)

Returns the triangular area between the three float triples.

float **poly_area**(defpoly*, float* middle)

Returns the area of the specified polygon given the float triple midpoint.

float **bend_poly_area**(defbendpoly*)

Returns the area of the specified bent polygon.

float **text_poly_area**(defpolyt*, float* middle)

Returns the area of the specified textured polygon given the float triple midpoint.

int **shatter_break_or_move**(Force*, float* distance, area)

Determines the result of the specified force, traveling the specified distance, and hitting a polygon of specified area.

void **add_the_vertices**(defpoly*, float* A, B, middle)

Adds the three vertex float triples to the polygon.

void **text_add_the_vertices**(defpolyt*, float* A, B, middle)

Adds the three vertex float triples to the textured polygon.

OPCODE ***triangle_into_two**(object*, opcode*)

Replaces the specified object's triangular polygon with two smaller triangular polygons. Returns the memory allocated to the polygon.

OPCODE ***text_triangle_into_two**(object*, opcode*)

Replaces the specified object's textured triangular polygon with two smaller textured triangular polygons. Returns the memory allocated to the textured polygon.

OPCODE ***shatter_triangulate**(object*, opcode*)

Replaces the specified object's triangular polygon with many smaller triangular polygons. Returns the memory allocated to the polygon.

OPCODE ***text_shatter_triangulate**(object*, opcode*)

Replaces the specified object's textured triangular polygon with many smaller textured triangular polygons. Returns the memory allocated to the textured polygon.

OPCODE ***bend_triangulate**(object*, opcode*, float* middle)

Replaces the specified object's polygon with a bent polygon. Returns the memory allocated to the polygon.

void **apply_bend_force_to_poly**(defbendpoly*, force*,
float* distance, area,deltatime)

Modifies the specified bent polygon's shape after a force hits it.

void **move_poly_in_the_direction_of_explosion**(poly*, force*,
float* direction,distance, area,deltatime)

Modifies the specified polygon's position after a force hits it.

void **move_textpoly_in_the_direction_of_explosion**(defpolyt*, force*,
float* direction,distance, area,deltatime)

Modifies the specified textured polygon's position after a force hits it.

OPCODE ***apply_force_to_poly**(object*, opcode*, force*, float* deltatime)

Breaks, bends or moves the specified polygon.

OPCODE ***apply_force_to_textpoly**(object*, opcode*, force*, float* deltatime)

Breaks, bends or moves the specified polygon.

void **apply_deforming_force_to_object**(object*, force*, float* deltatime)

Breaks, bends or moves the specified object.

void **continue_explosion**(object*, float* deltatime)

Continues the explosion process on the object after the specified force has been removed.

E. NON-DEFORMING FORCE FUNCTIONS

void **update_object_physics**(object*,summove*, float* deltatime)

Calculates the updated world frame of reference position, orientation, velocities, and accelerations of the specified object.

void **apply_non_deforming_force_to_object**(object*, force*,summove*)

Calculates the affect (linear and radial) of the specified local force on the specified object.

void **apply_global_force_to_object**(object*, force*,summove*, float* deltatime)

Calculates the affect (linear) of the specified global force on the specified object.

APPENDIX C
NPS OFF MOVER TOOL
USERS GUIDE

F. Introduction

The objective of the manual is to provide a broad overview of the nature and capabilities of the Mover Tool, plus a specific "how to" guide to rapidly orient both the novice and advanced user. The contents of this manual can also be found in the help subsystem of the Mover Tool.

1. Organization

a. Files

All files for the Mover tool executable exist in the ~zyda/rdobj directory. It is advised to place this directory in your path and/or adding an alias to run the Mover Tool (mover).

2. Hardware

The Mover Tool is compatible on all Silicon Graphics (SGI) Iris workstations. Advanced OFF features requiring VGX hardware (such as textured surfaces) are supported. The interface was designed with the NPS Panel Designer and utilizes several virtual controls, also known as "widgets" which simulate mechanical dials, sliders, and push buttons. The only required input devices are a mouse and keyboard.

a. Mouse

The mouse is used to manipulate the virtual controls in the various panels of the user interface. A particular control is activated by positioning the mouse cursor over it and pressing the LEFT mouse button. The MIDDLE mouse button, when pressed with the LEFT button, lowers the sensitivity of the selected actuator. This facility is useful when "fine-tuning" an actuator's value. The RIGHT mouse button activates a simple pulldown menu with two selections: HELP subsystem and EXIT the Mover Tool.

Warning: the EXIT selection **does not** prompt for confirmation!

b. Keyboard

The keyboard is used to enter descriptive names for an object, force and OFF file names.

3. Navigating around the Tool's panels

The Mover Tool's interface consists of a main viewing window with the controls in several panels along the left side, (Figure C.1).

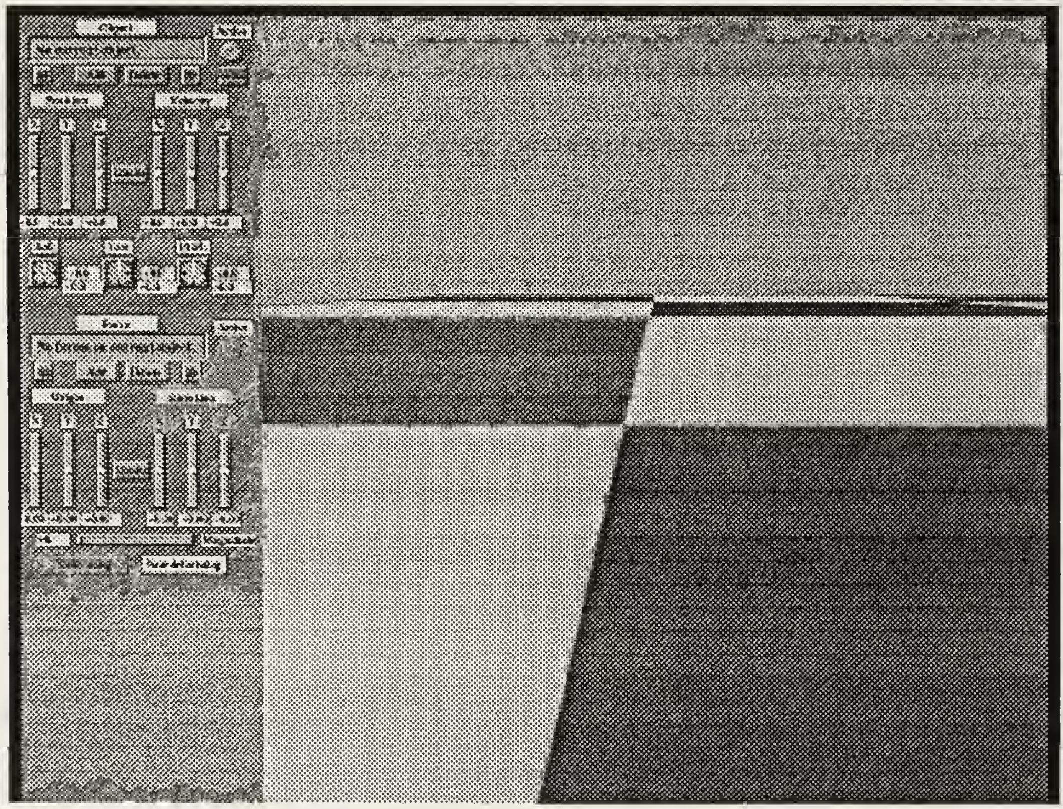


Figure C.1 Full Screen

At start-up, the only control panels visible are the OBJECT, FORCE and ACTION panels.

a. Main Viewing Window

This is the largest window in the middle of the screen, (Figure C.2).

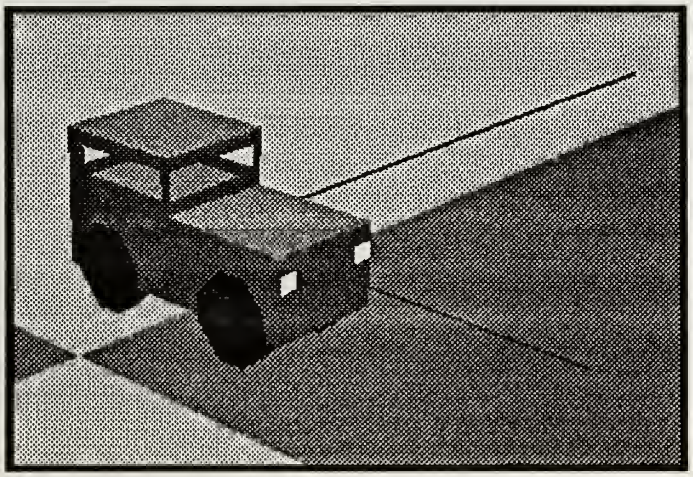


Figure C.2 Main Viewing Window

Initially, the only object visible is a checkerboard surface of gray and white one hundred thousand square meter tiles. This uniform ground feature was selected over a textured surface to facilitate the visual quantification of an object's size, acceleration and velocity.

b. OBJECT Panel

This panel is in the upper left corner, (Figure C.3)

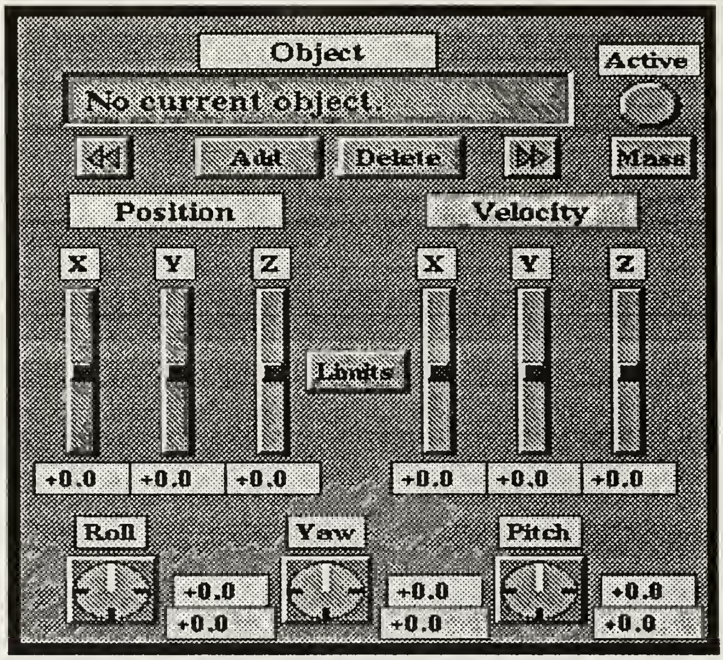
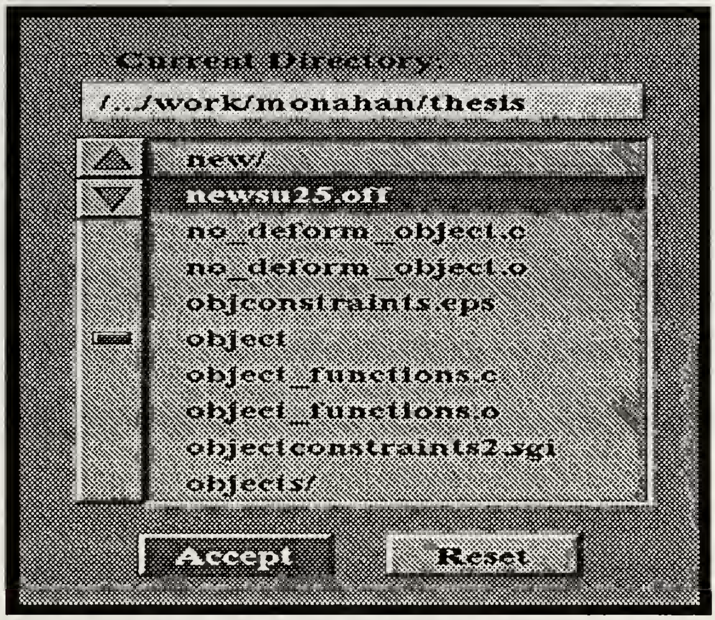


Figure C.3 Object Panel

The panel's controls start with a typein that displays the name of the "currently selected" object. Initially, the typein displays the fact that there are no objects in the environment. As each object file is added, the typein displays the name of the file containing the object. The user can change the name by clicking on the typein, backspacing over each incorrect letter or pressing the DELETE key to clear the typein, and then entering the new name followed with the ENTER/RETURN key.

To the right of the typein is a toggle button indicating the object's status. If highlighted, the object is ACTIVE and under the influence of all ACTIVE forces attached to it. An INACTIVE object remains at equilibrium and its motion is described by its inertia.

Below is a row of four object manipulator buttons. The ADD button displays a scrollable directory browser, (Figure C.4)



Current Directory:
/../work/monahan/thesis

```
new/
newsu25.off
no_deform_object.c
no_deform_object.o
objconstraints.eps
object
object_functions.c
object_functions.o
objectconstraints2.sgi
objects/
```

Figure C.4 Directory Browser

Highlighting an OFF filename with the LEFT mouse button and then clicking on the ACCEPT button displays the object in the environment and removes the browser. Scrolling the current directory up or down is accomplished with the arrow keys on the keyboard, clicking on the scrollbar arrows to the left of the listing, or by clicking and dragging the "foot" or highlighted dark area within the scrollbar. Changing directories is accomplished by highlighting a directory name and then clicking on the directory listing ACCEPT button.

The DELETE button on the OBJECT panel removes the currently selected object. The buttons to either side of ADD/DELETE traverse the list of environment objects. The button on the left labeled "<<" selects the previous object while the button on the right labeled ">>" selects the next object. To the right is the MASS button that activates the MASS ATTRIBUTES panel with vertical sliders which control the current object's MASS AMOUNT, CENTER of MASS, and ELASTICITY, (Figure C.5)

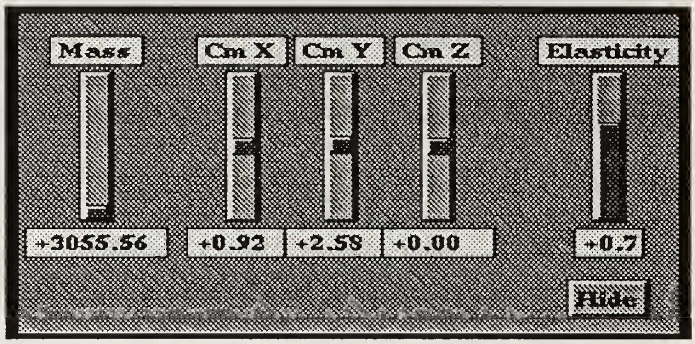


Figure C.5 Mass Attributes Panel

Below is a row of six vertical linear position and velocity sliders. The default boundary values for the setting of velocity and position are +/- 10,000 meters and meters/second respectively. In the midst of these sliders is the CONSTRAINTS button that activates the OBJECT CONSTRAINTS panel with vertical sliders which control the linear/ angular position and velocity boundary values, (Figures C.6a and C.6b)

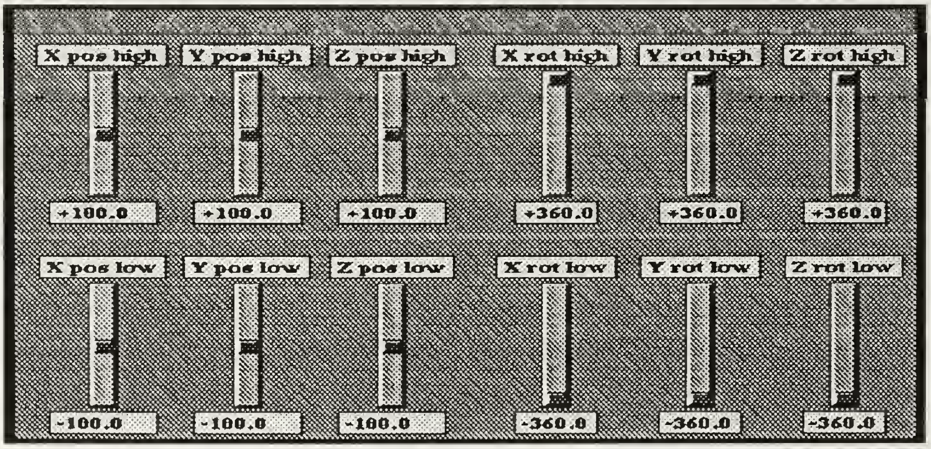


Figure C.6a Object Constraints Panel (left)

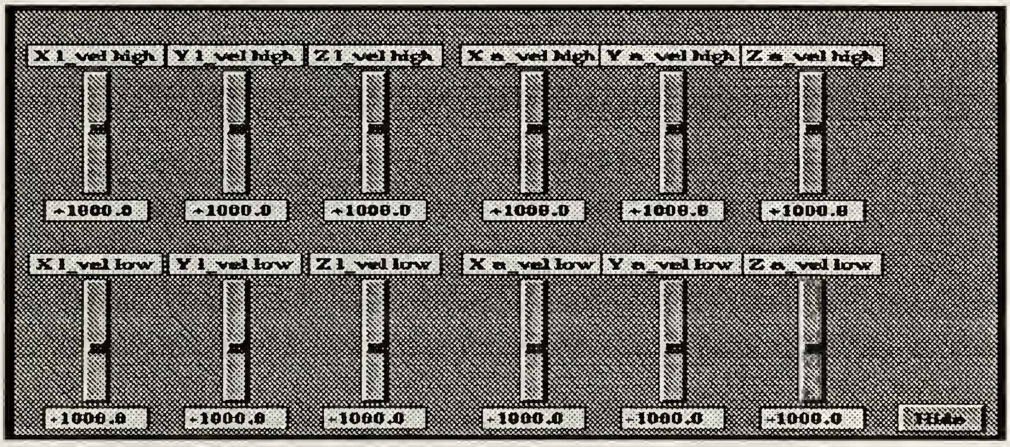


Figure C.6b Object Constraints Panel (right)

The HIGH sliders set the upper limits for possible position, rotation, linear and angular velocity values. The LOW sliders set the respective lower limits. Below is a row of three dials controlling ROLL, PITCH and YAW. To the right of each dial are two digital repeaters that indicate the current object's angular position and velocity.

c. *FORCE Panel*

This panel is below the object panel, (Figure C.7).

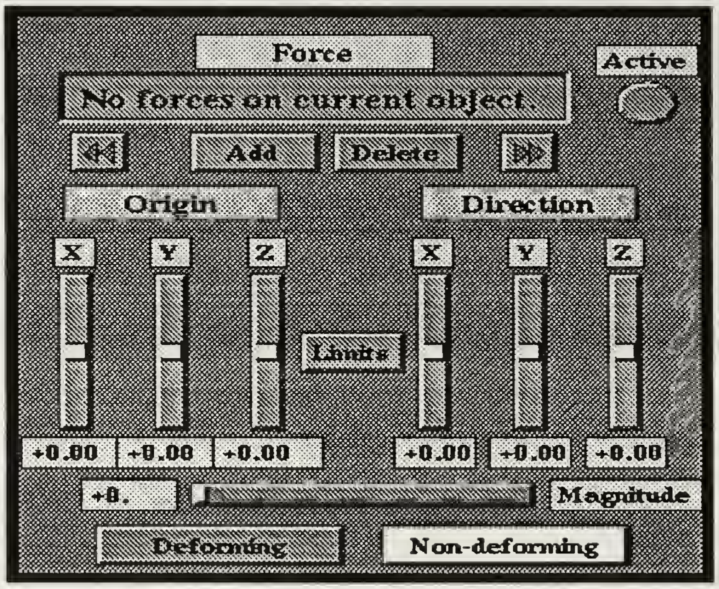


Figure C.7 Force Panel

The panel's controls start with a typein that displays the name of the "currently selected" force. Initially, the typein displays the fact that there are no forces attached to the current object. As each force is added, the typein indicates that the force is yet unnamed. The user can enter a new name or change an existing name by clicking on the typein, backspacing over each incorrect letter or pressing the DELETE key to clear the typein, and then entering the new name followed with the ENTER/RETURN key.

To the right of the typein is a toggle button indicating the force's status. If highlighted, the force is ACTIVE and will influence the object to which it is attached. An INACTIVE force is in a state of suspended animation.

Below is a row of four force manipulator buttons. The ADD button adds a default force to the list of forces attached to the current object. The DELETE button removes the currently selected force.

The buttons to either side of ADD/DELETE traverse the list of attached forces. The button on the left labeled "<<" selects the previous force while the button on the right labeled ">>" selects the next force. The "current" NON-DEFORMING force is displayed as a GREEN vector while the remaining forces are BLUE. DEFORMING forces are always displayed as a RED 3D snowflake.

Below is a row of six vertical sliders for changing the force's origin and direction. The default boundary values for the setting of the force's origin values are +/- 10,000 meters. In the midst of these sliders is the CONSTRAINTS button that activates the FORCE CONSTRAINTS panel with vertical sliders which control the force origin and magnitude boundary values (Figure C.8).

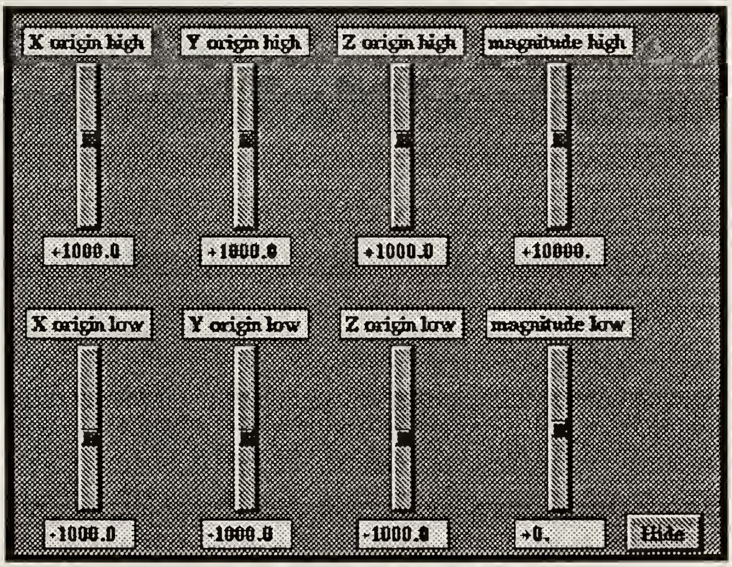


Figure C.8 Force Constraints Panel

The HIGH sliders set the upper limits for possible origin and magnitude values. The LOW sliders set the respective lower limits. Next is a horizontal slider that sets the force magnitude. The last controls in the panel are two toggle buttons that indicate the type, DEFORMING or NON-DEFORMING, of the force.

d. *ACTION Panel*

This panel is in the lower left corner, (Figure C.9)

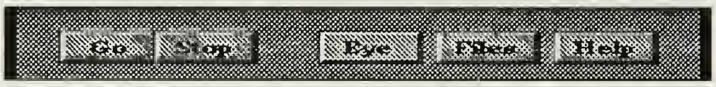


Figure C.9 Action Panel

The first two buttons control the environment's animation. Highlighting the GO button puts into motion all active objects with active forces. Removing the button's highlight halts the animation process. Pressing the RESET button resets the current object's linear/angular positions and velocities. The third button, EYE, activates a new panel that allows the user to control eye position and direction. The fourth button, FILES, activates a new panel that allows the user to save the current object back to an OFF file, save all objects in the environment to an OFF theater file (group of files), and add the objects in an OFF theater file to the environment. The last button, HELP, activates a scrollable window with the contents of this manual.

(1) EYE Panel -This panel appears to the right of the ACTION panel (Figure C.10).

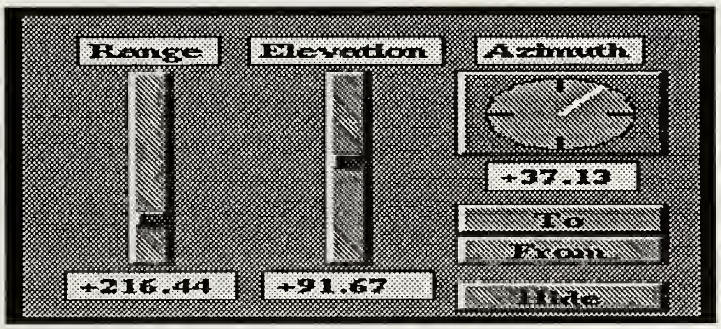


Figure C.10 Eye Panel

This panel has controls for eye range, azimuth, and height from a specified point. The default settings place the viewpoint looking from a point 100 meters radius and 20 meters height from the center of the tiled floor. The user may select to shift the focus TO the currently selected object, with the control parameters relative from the object's center. The user may also select to view FROM the current object in the direction of the control parameters. If an object viewpoint is not specified in the OFF file, a default viewpoint centered 10% above the object's height is used.

(2) FILE Panel -This panel appears above the ACTION panel (Figure C.11).

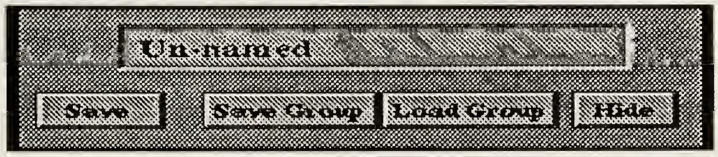


Figure C.11 File Panel

This panel includes a typein to specify the saved OFF object or OFF theater filename. If a unique filename is not specified, the user is notified that an existing file will be overwritten and is given the option to cancel the save operation. If one or more objects have the same name and the user elects to save all objects into a theater file, the user is notified that a filename conflict exists and that the last object with the duplicate name will overwrite the other objects with the duplicate name. Clicking the LOAD GROUP button to load an OFF theater file, displays a scrollable listing of the current directory. The procedure for selecting a theater file is identical to that for selecting an OFF object file.

4. How to Modify a Sample OFF File

a. Adding, Deleting, Selecting Objects

At the command line of an IRIS in the Graphics Lab, type "mover". The OBJECT, FORCE and ACTION panels appear to the left of the main viewing window. In the OBJECT panel, click on the ADD button. The directory browser appears displaying the contents of the current directory. Highlight an OFF file of interest and click on the ACCEPT button. The directory browser disappears and the OFF object appears in the center of the main viewing window. Click on the typein at the top of the OBJECT panel and press the DELETE key. Enter another meaningful name for the OFF object, followed with a ENTER/RETURN. Add another OFF object in a similar fashion. Use the "<<" and ">>" keys on the OBJECT panel to select the current object. Press the DELETE button and that object disappears. Use the ADD button to add a couple more objects.

b. Object Attributes

(1) Mass - Select one of the objects and press the MASS button. The MASS panel appears to the right. The panel will display default mass values if the OFF object did not have any previous mass data. Adjust the MASS slider to reflect a mass of

5,000 kilograms and adjust the ELASTICITY slider to reflect an energy absorption characteristic of 0.8. Click on the HIDE button when done.

(2) Constraints - Click on the object LIMITS button on the OBJECT panel. The OBJECT CONSTRAINTS panel appears to the right. Adjust the Y POS (position) sliders to constrain the object's range of vertical movement from 0.0 to 1000.0 meters. Adjust the X L_VEL (linear velocity) sliders to constrain the object's range of fore/ aft velocity from -10.0 to 50.0 meters/second.

(3) Position and Orientation - Adjust the three position sliders and three ROLL, YAW, PITCH dials to position and orient the object as desired.

(4) Velocity - Adjust the three velocity sliders to specify the object's initial linear velocity as desired.

c. Adding, Deleting, Selecting Forces

Click on the ADD button on the FORCE panel. A default force is appended to the current object's force list. Click on the typein at the top of the FORCE panel and press the delete key. Enter another meaningful name for the force, followed with a ENTER/RETURN. Add another force in a similar fashion. Use the "<<" and ">>" keys on the OBJECT panel to select the "current" force. Press the DELETE button and that force is removed from the force list. Use the ADD button to add a couple more forces.

d. Non-deforming Force Attributes

(1) Constraints - Click on the force LIMITS button on the FORCE panel. The FORCE CONSTRAINTS panel appears to the right. Adjust the X ORIGIN sliders to constrain the force's range of horizontal movement from -10.0 to 0.0 meters. Adjust the MAGNITUDE sliders to constrain the force's range of magnitude from 0.0 to 5000.0 newtons.

(2) Origin and Direction - Adjust the three origin sliders to position (-5.0, 0.0, 0.0). This is a point of affect 5.0 meters behind the current object. Adjust the three direction sliders to direction (1.0, 0.0, 0.0). This force direction will push the object forward.

(3) Magnitude - Adjust the magnitude slider to specify 500.0 newtons. Notice the vector adjacent to the object.

(4) Type - Click on the NON-DEFORMING button.

e. Point of View

Click on the EYE button on the ACTION panel. The EYE panel appears to the right. Click on the TO button as we may want to keep the current object in the center of the main viewing window. Select a comfortable viewpoint, 50.0 meters out and 10.0 meters up. Our eye will maintain this relative position from the object. Click on the FROM button and notice that **neither** button is highlighted. This viewpoint is measured from the environment's center. Click on the FROM button again, as we may want to view the environment from the object's point of view. Now the sliders specify a point in space from the object's viewpoint. Go back to the TO position and click on the HIDE button when done.

f. Non-deforming Movement

Click on the GO button on the ACTION panel. The object will start to accelerate forward, slowly increasing in velocity. Adjusting the force parameters will result in different resultant motion. Clicking on the GO button again stops the animation. Clicking on the RESET button, sets the current object's position, orientation and velocities to their default values.

g. *Deforming Force Attributes*

Delete all forces on the current object. Add a new force in the prescribed manner, positioning it at a point coincident with the object's center (0.0, 0.0, 0.0) and click on the DEFORMING button.

h. *Deforming Movement*

Click on the GO button on the ACTION panel. The polygons within the object will start to shatter and explode, assuming that the polygons' breaking value is low enough for the force applied. Clicking on the GO button again stops the explosion. Clicking on the RESET button, sets the current object's position, orientation and velocities to their default values. We cannot "re-assemble" a destroyed object as yet. The alternative is to replace the damaged object with an undamaged copy by deleting the destroyed object and replacing it from disk.

i. *Saving Objects and Groups of Objects*

After you have specified an object and the attached forces, you will want to save it for future use. Click on the FILES button on the ACTION panel. The FILES panel appears to the right. Click on the typein at the top of the FILES panel and press the DELETE key. Enter another meaningful name for the OFF filename, followed with a ENTER/RETURN. Then click on the SAVE button. Click on the SAVE button again. Since the filename in the typein currently exists, you will see a message indicating that the current object will overwrite the existing file. You can confirm or cancel this operation.

You can also save several objects to an OFF theater file in a similar method. Specify an OFF theater filename and then click on the SAVE GROUP button. If any of the objects in the theater file will have the same name, you will see a message indicating that the file with the duplicate object name will contain the last object with the duplicate name.You can confirm or cancel this operation.

LIST OF REFERENCES

(Barr 1987)
A. H. Barr, "Dynamic Constraints", *ACM SIGGRAPH '87 Tutorial Notes: Topics in Physically-Based Modeling,* 1987

(Barr 1988)
A. H. Barr, "Teleological Modeling", *ACM SIGGRAPH '88 Course Notes #27: Developments in Physically-Based Modeling, Section E*, August, 1988

(Barzel 1988a)
R. Barzel and A. H. Barr, "A Modeling System Based on Dynamic Constraints," *ACM SIGGRAPH '88 Conference Proceedings,* Vol 22 No 4, August 1988, pp. 179-188

(Barzel 1988b)
R. Barzel and A. H. Barr, "Controlling Rigid Bodies with Dynamic Constraints," *ACM SIGGRAPH '88 Course Notes #27: Developments in Physically-Based Modeling, Section E*, August, 1988

(Brett 1987)
C. Brett, S. Pieper, D. Zeltzer, "Putting it all Together: An Integrated Package for Viewing and Editing 3D MicroWorlds," *Proc. 4th Usenix Computer Graphics Workshop*, October 1987

(Goldstein 1980)
H. Goldstein, *Classical Mechanics*, Second Edition, Addison-Wesley, Reading, MA, 1980

(Jurewicz 1989)
T. Jurewicz, "A Real Time Autonomous Underwater Vehicle Dynamic Simulator," M.S. Thesis, Naval Postgraduate School, Monterey, CA, June 1989

(Spiegel 1988)
M. Spiegel, "*Applied Differential Equations, Third Edition*," Prentice Hall, Inc., Englewood Cliffs, N.J. 1988, pp. 1-26

(Sturman 1989)
D. Sturman, D. Zeltzer, and S. Pieper, "The Use of Constraints in the bolio System," *ACM SIGGRAPH '89 Course Notes: Implementing and Interacting with Realtime Microworlds*, Boston, MA, July 31, 1989

(Wilhelms 1986)
J. Wilhelms, "Virya - A motion Control Editor for Kinematic and Dynamic Animation," *Proceedings of Graphics Interface 86*, May, 1986, pp. 141-146

(Wilhelms 1987)
J. Wilhelms, “Using Dynamic Analysis for Realistic Animation of Articulated Bodies,” *IEEE Computer Graphics and Applications*, Vol 7 No 6, June 1987, pp. 12-27

(Wilhelms 1988)
J. Wilhelms, “Dynamics for Computer Graphics: A Tutorial,” *Computing Systems*, USENIX Association, Winter, 1988, pp. 63-93. also *UCSC Computer and Information Science Technical Report* UCSC-CRL-87-5

(Wilhelms 1990)
J. Wilhelms and R. Skinner, “An Interactive Approach to Behavior Control,” *ACM SIGGRAPH ‘90 Course Notes: Developments in Physically-Based Modeling*, July, 1990

(Zeltzer 1989)
D. Zeltzer, S. Pieper, and D. Sturman, “An Integrated Graphical Simulation Platform,” *Proceedings of Graphics Interface 89*, June 19-23, 1989, London, Ontario, p. 266-274

(Zyda 1991a)
M. Zyda, “Book 7, Computer Graphics”, *Naval Postgraduate School Course Notes CS 4202: Computer Graphics*, 2 April 1991

(Zyda 1991b)
M. Zyda, “Book 9, Computer Graphics”, *Naval Postgraduate School Course Notes CS 4202: Computer Graphics*, 31 May 1991

(Zyda 1991c)
M. Zyda and D. Pratt, “NPSNET: A 3D Simulator for Virtual World Exploration and Experimentation”, *Society for Information Display 1991 International Symposium Digest of Technical Papers*, 31 May 1991

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Dr. Michael J. Zyda Naval Postgraduate School Code CS, Department of Computer Science Monterey, CA 93943-5100	8
4.	David Pratt Naval Postgraduate School Code CS, Department of Computer Science Monterey, CA 93943-5100	2
5.	Lieutenant James G. Monahan 1040 Halsey Drive Monterey, CA 93940	1

Thesis
M6726 Monahan
c.1 NPSNET.